REMARKS

Claims 1 and 4 have been amended to more fully point out and distinctly claim the subject matter of the present invention. Thus, claims 1-12 remain pending in the present application. In view of the foregoing amendments and the following remarks, it is respectfully submitted that all of the presently pending claims are allowable.

I. The Rejections Under 35 U.S.C. § 101 Should Be Withdrawn

The Examiner has rejected claims 1-7 under 35 U.S.C. 101 as being directed to non-statutory subject matter. *Office Action*, page 2. The applicants respectfully disagree with the Examiner's statements regarding the form of the claims and whether the claims as written are non-statutory. However, in the interest of expediting the allowance of the present claims, the applicants have amended independent claims 1 and 4 to recite a system for performing an operation on a processor (claim 1) and a class loader present in the form of software code that is operable by a processor (claim 4). Accordingly, the applicants respectfully submit that claims 1-7 are directed to statutory subject matter and the rejection under 35 U.S.C. 101 should be withdrawn.

II. The Rejections Under 35 U.S.C. § 102 Should Be Withdrawn

The Examiner has rejected claims 1-12 under 35 U.S.C. 102(e) as being anticipated by U.S. Patent 6,542,887 to Abbott (the "Abbott reference"). *Office Action*, page 4. The Abbott reference describes a Java Virtual Machine ("JVM") system for retrieving native code libraries stored within archived files of the class path. *Abbott reference*, col. 2, lines 28-42. Native code libraries contain machine specific code and are usually located on a particular machine. Class code is platform independent and may be stored in a plurality of locations, such as in archived files (e.g., zip, arj, etc.) which are located in the directories of the class path. *Id.*,

col. 1, lines 32-33. As the JVM executes, it loads native code libraries from a library path and class files from the class path. *Id.*, col. 3, lines 4-8. The Abbott reference augments the native code library load mechanism to search the class path and the archived files stored therein for the required native libraries. *Id.*, col. 6, lines 9-13. After the the native code library is located in the class path, it is extracted from the archived file and copied to the cache directory. This allows the library loading routine to "continue in a conventional manner, using its normal resolution and loading mechanism to load the library from the cache directory." *Id.*, col. 3, lines 48-51.

Claim 1 of the application recites a system comprising: "a stream source class loader retrieving streaming data to create a desired class object" and "an interface coupled to the stream source class loader" in combination with "a plurality of streaming sources containing information including the location of data, wherein requests for data are communicated from the stream source class loader to the streaming sources via the interface and, data passes from the stream sources to the stream source class loader via the interface, the streaming sources searching the data locations for the requested data."

In contrast, the Abbott reference merely discloses a system for locating native libraries and not loading classes from a plurality of streaming sources. There is no teaching or suggestion in the Abbott reference as to loading classes from anywhere other than the classpath. The Abbott reference does not teach or suggest a system comprising of "a stream source class loader" and "an interface" in combination with "a plurality of streaming sources." The Abbott reference only discusses the conventional load mechanisms without explaining how classes from multiple streaming sources may be instantiated. For instance, after a file is located "the *conventional* native code library load mechanism . . . can continue in the *conventional* manner, using its normal resolution and loading mechanism to load the library." *Abbott reference*, col. 3, lines 9-51. The present invention describes in detail the structure and the functionality of the

stream source class loader. More specifically, the stream source class loader may "load classes from any one of stream sources 120-140 without the need to implement additional custom class loaders." *Specification*, page 5, ¶ 12. Thus, the present invention, unlike the Abbott reference does not rely on the conventional class loading mechanisms and discloses a custom stream source class loader suitable for loading multiple classes from a plurality of stream sources that may be outside the classpath.

The Abbott reference also fails to disclose "an interface coupled to the stream source class loader" as recited in claim 1. As discussed above the Abbott reference relies on conventional class loading mechanisms and does not teach "an interface to allow stream source class loader 110 to access the class byte code as streaming data available via stream sources 120-140." *Specification*, page 6, ¶ 13. The Examiner incorrectly equates the interface as recited in claim 1 with a path shown in Fig. 1 of the Abbott reference. That particular path between the class loader and Class path and Libpath is not labeled nor described in the Abbott reference. The path merely denotes the search direction of the library load mechanism as illustrated in the accompanying specification. Conversely, the interface of the present application reads the byte data from sources and delivers it to the stream source class loader. In addition, the interface is also used to transmit requests for data from the stream source class loader to the stream sources. *Specification*, page 6, ¶ 13. As recited in claim 1, "requests for data are communicated from the stream source class loader to the streaming sources via the interface."

Accordingly, for at least the reasons described above, the applicants respectfully submit that the Abbott reference neither teaches nor discloses "a stream source class loader retrieving streaming data to create a desired class object" and "an interface coupled to the stream source class loader" wherein "requests for data are communicated from the stream source class loader to the streaming sources via the interface and, data passes from the stream sources to the

stream source class loader via the interface" as recited in claim 1. Applicants therefore respectfully request that the rejection of claim 1 and all the rejected claims dependent therefrom (claims 2 and 3) be withdrawn.

Similarly, the remaining independent claims have similar limitations as described above with reference to claim 1. Specifically, claim 4 recites a system comprising of "a receiving module to receive a request for a desired class object" and *"an interface module which receives requests for data from the receiving module and retrieves the streaming data* from the stream source module."

Claim 8 recites a method of loading a class object, comprising the steps of: "*receiving a load request for the class object by a class loader*" and "*passing the load request to an interface module*, wherein the interface module includes a method to retrieve streaming data associated with the class object."

Accordingly, for the same reasons as described above with reference to claim 1, the applicants respectfully request that the rejection of claims 4 and 8 and the claims depending therefrom (claims 5-7 and 9-12) be withdrawn.

CONCLUSION

In view of the remarks submitted above, the applicants respectfully submit that the present case is in condition for allowance. All issues raised by the Examiner have been addressed, and a favorable action on the merits is thus earnestly requested.

Respectfully submitted,

Dated: March 9, 2004

By: _____
Michael J. Marcin (Reg. No. 48,198)

FAY KAPLUN & MARCIN, LLP
150 Broadway, Suite 702
New York, NY 10038
(212) 619-6000 (phone)
(212) 208-6819 (facsimile)